



UNIVERSITÀ
DEGLI STUDI
DI PADOVA



DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

UNIVERSITÀ DEGLI STUDI DI PADOVA

TRIENNALE DI INGEGNERIA INFORMATICA

CORSO DI PROGRAMMAZIONE DI SISTEMI EMBEDDED

Sviluppo di una app fotocamera in Android tramite Libreria CameraX

Relazione e progetto di:

Nicola Busato 2009663

Nicolas Brentel 2009673

Tommaso Leoni 2009666

Anno Accademico 2022/2023

Indice

1	Introduzione	1
2	Classe Camera	2
2.1	Compatibilità	2
2.1.1	Camera Permission	2
2.1.2	Camera Features	2
2.1.3	Ulteriori autorizzazioni	3
2.2	Funzionalità	3
2.2.1	Utilizzare un Intent	3
2.2.2	Accedere alla Camera	4
2.2.3	Gestire i Camera Parameters	4
2.2.4	Creare un'anteprima	5
2.2.5	Scattare una foto	7
2.2.6	Registrazione un video	8
2.2.7	Impostare i Camera Parameters	8
2.3	Pro e Contro	9
2.3.1	Pro	9
2.3.2	Contro	10
2.3.3	Conclusioni	10
3	Classe Camera2	11
3.1	Compatibilità	11
3.2	Funzionalità	11
3.2.1	CameraCaptureSession e CaptureRequest	11
3.2.2	HDR video	12
3.2.3	Multi-camera API	13
3.2.4	Controllo manuale dei parametri	14
3.3	Pro e Contro	19
3.3.1	Pro	19
3.3.2	Contro	19
3.3.3	Conclusioni	20
4	Classe CameraX	21
4.1	Introduzione	21
4.1.1	Versioni	21
4.1.2	Creazione	21
4.1.3	App di terze parti	22
4.1.4	Funzionamento e estensioni	22
4.2	Architettura	23

4.2.1	Struttura	23
4.2.2	API model	24
4.2.3	Ciclo di vita	25
4.2.4	Permessi	26
4.2.5	Dipendenze	26
4.3	Configurazione	26
4.3.1	Panoramica	26
4.3.2	Configurazione automatica	27
4.3.3	Rotazione	27
4.3.4	Crop rect	28
4.3.5	Selezione Camera	29
4.3.6	Multi Camera	29
4.3.7	Risoluzione	29
4.3.8	Zoom	29
4.3.9	Flash	30
4.3.10	Focus and Metering	30
4.4	Pro e Contro	30
4.4.1	Pro	30
4.4.2	Contro	31
4.4.3	Conclusioni	32
5	Conclusioni	33
5.1	Confronto generale	33
5.2	Perchè abbiamo scelto CameraX	33
	Bibliografia	35

1 Introduzione

Questo documento mira ad introdurre le funzionalità e le caratteristiche delle tre API per la gestione della fotocamera sviluppate da Google per la piattaforma Android: **Camera**, **Camera2** e **CameraX**. I capitoli successivi trattano le librerie in ordine cronologico di rilascio.

Camera è la prima, introdotta con *Android 1.0*, è quella compatibile con il maggior numero dispositivi, ma è deprecata.

Camera2 è stata rilasciata a partire da *Android 5.0*, come evoluzione della precedente, e offre funzionalità più avanzate e migliori opzioni di personalizzazione.

CameraX, rilasciata anch'essa a partire da *Android 5.0*, si pone l'obiettivo di unificare lo sviluppo di app per la fotocamera su dispositivi Android provenienti da diversi produttori e dotati di versioni diverse del sistema operativo.

Per ulteriori approfondimenti si faccia riferimento allo specifico capitolo di ciascuna libreria.

Il capitolo **Conclusioni** infine contiene un confronto tra le diverse API e spiega le scelte fatte per lo sviluppo dell'applicazione presentata assieme a questo report, che possono essere un buon punto di partenza per aiutare gli sviluppatori a decidere quale libreria utilizzare.

2 Classe Camera

Camera è stata la prima API sviluppata per sfruttare l'hardware fotocamera dei dispositivi Android e per sviluppare un'ampia gamma di applicazioni basate sulla visione. A partire dall'*API level 21* è però diventata obsoleta ed è consigliato invece l'utilizzo dell'**API CameraX**.

2.1 Compatibilità

L'API Camera è stata introdotta in *Android 1.5 (API level 3)*. La prima versione comprendeva le funzionalità base della fotocamera come mostrare un'anteprima in tempo reale e catturare e riprodurre immagini.

A partire da *Android 2.0 (API level 5)* sono state introdotte nuove features, come il focus e lo zoom, assieme anche al supporto per gestire le fotocamere anteriori dei dispositivi.

Infine, con *Android 2.3 (API level 9)*, è stato introdotto il supporto per l'utilizzo di fotocamere multiple e per la registrazione di video.

2.1.1 Camera Permission

È necessario richiedere l'autorizzazione per utilizzare l'hardware della fotocamera e le Camera features nel Manifest file.

```
1 <uses-permission android:name="android.permission.CAMERA" />
```

Questa autorizzazione implica anche che l'app utilizzi la feature **android.hardware.camera**, a meno che non venga specificato diversamente nel Manifest con questa scrittura:

```
1 <uses-permission android:name="android.permission.CAMERA" android:required="false" />
```

2.1.2 Camera Features

Se l'applicazione che si vuole sviluppare richiede obbligatoriamente che sia installata una camera sul dispositivo, dovremo specificare anche quali camera features andremo ad utilizzare nel documento nel Manifest file.

Al tempo della scrittura di questo documento le feature disponibili sono:

- **android.hardware.camera.any**: questa feature specifica l'utilizzo di una fotocamera esterna, indipendentemente dal suo posizionamento sul dispositivo
- **android.hardware.camera**: questa feature specifica l'utilizzo della fotocamera posteriore (detta anche *world-facing*) del dispositivo
- **android.hardware.camera.front**: questa feature specifica l'utilizzo della fotocamera anteriore (detta anche *user-facing*) del dispositivo
- **android.hardware.camera.external**: questa feature specifica l'utilizzo di una fotocamera esterna, connessa al dispositivo dall'utente, anche se non ne garantisce la presenza

- **android.hardware.camera.autofocus**: questa feature specifica l'utilizzo dell'autofocus feature supportata dalla fotocamera del dispositivo
- **android.hardware.camera.flash**: questa feature specifica l'utilizzo della flash feature supportata dalla fotocamera del dispositivo
- **android.hardware.camera.level.full**: questa feature specifica l'utilizzo del FULL level di cattura immagini supportata da almeno una delle fotocamere del dispositivo

Per visualizzare una lista aggiornata fare riferimento a **Camera Features**.

2.1.3 Ulteriori autorizzazioni

Oltre alle autorizzazioni strettamente legate all'hardware della fotocamera potrebbe essere necessario richiedere alcune delle seguenti, in base ai casi d'uso che si vogliono implementare.

È possibile **salvare immagini e video nella memoria esterna (SD card)** per i dispositivi *Android* (*API level 29*) o inferiore, specificando l'autorizzazione nel Manifest file.

```
1 <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Per poter **registrare l'audio nei video** è necessario richiedere l'autorizzazione per registrare audio nel Manifest file.

```
1 <uses-permission android:name="android.permission.RECORD_AUDIO" />
```

Se si vuole **contrassegnare le foto con la posizione dove sono state scattate** è possibile utilizzare il sistema GPS. Nel caso in cui si stia sviluppando per *Android 5.0* (*API level 21*) o superiore è necessario richiedere l'autorizzazione nel Manifest file.

```
1 <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
2 ...
3 <!-- Needed only if your app targets Android 5.0 (API level 21) or higher. -->
4 <uses-feature android:name="android.hardware.location.gps" />
```

2.2 Funzionalità

2.2.1 Utilizzare un Intent

Se nell'applicazione che si vuole creare si vogliono solo gestire immagini e video, ma non occuparsi della loro creazione, è possibile delegare questo compito attraverso un **Intent** all'applicazione di sistema predefinita.

```
1 // Creazione dell'intent per avviare l'app fotocamera predefinita
2 var takePictureIntent = Intent(MediaStore.ACTION_IMAGE_CAPTURE);
3 // Se l'app e' disponibile (esiste) si avvia la cattura dell'immagine
4 if (takePictureIntent.resolveActivity(getPackageManager()) != null) {
5     startActivityForResult(takePictureIntent, REQUEST_IMAGE_CAPTURE);
6 }
7 ...
```

```

8 // Il result potrà essere gestito dl metodo callback
9 override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
10     super.onActivityResult(requestCode, resultCode, data)
11     // Controllo se l'Intent e' stato richiamato per la cattura della foto
12     if (requestCode == REQUEST_IMAGE_CAPTURE && resultCode == Activity.RESULT_OK) {
13         // La foto e' stata catturata con successo, puoi gestirla qui
14         // data contiene i dati dell'immagine, ad esempio il percorso del file
15     } else {
16         // La cattura della foto e' stata annullata o ha generato un errore
17     }
18 }

```

2.2.2 Accedere alla Camera

Se nell'applicazione ci si vuole occupare anche della creazione di immagini e video in maniera diretta è necessario accedere alla fotocamera del dispositivo. Una volta verificato che l'hardware sia disponibile è necessario richiedere l'accesso creando un'istanza della classe **Camera**.

Per accedere alla camera primaria del dispositivo si utilizza il metodo **Camera.open()**, che lancia una **RuntimeException** nel caso in cui il metodo sia già stato chiamato da un'altra applicazione. È quindi necessario che esista un solo oggetto di tipo Camera attivo alla volta per ciascuna fotocamera sul dispositivo.

Una volta che l'applicazione ha finito di utilizzare la camera deve essere chiamato il metodo **Camera.release()** per rilasciare la risorsa e renderla nuovamente disponibile alle altre applicazioni.

Nei dispositivi *Android 2.3 (API level 9)* o superiore è possibile accedere ad una specifica camera usando il metodo **Camera.open(cameraId: Int)**. Dove cameraId è un valore compreso tra 0 e il valore restituito dal metodo **getNumberOfCameras() - 1** (0 indica la prima fotocamera posteriore).

Note

- Se l'eccezione lanciata dal metodo **Camera.open()** non viene gestita il sistema terminerà l'applicazione
- In alcuni dispositivi il metodo **Camera.open()** richiede un lungo intervallo di tempo per essere completato, per questo motivo si consiglia di richiamarlo in un thread a parte, per evitare di bloccare il main thread

2.2.3 Gestire i Camera Parameters

Una volta ottenuto l'accesso alla fotocamera possiamo ottenere ulteriori informazioni sulle sue funzionalità usando il metodo **Camera.getParameters()** che ritorna un oggetto di tipo **Camera.Parameters** che rappresenta le impostazioni correnti della Camera.

Utilizzando i metodi di questo oggetto è possibile modificarlo per selezionare le impostazioni desiderate,

che possono poi essere caricate utilizzando il metodo `Camera.setParameters(parameters: Camera.Parameters)`.

Note

- È importante, prima di utilizzare uno dei metodi `Camera.Parameters#setXXX`, richiamare il metodo `Camera.Parameters#getSupportedXXX`, che ritornerà null nel caso in cui `XXX` non sia un'impostazione supportata dalla Camera corrente (dove `XXX` indica uno specifico Parameter).
- Quando si sviluppa per *API level 9* o superiore è possibile utilizzare il metodo `Camera.getCameraInfo()` per determinare la posizione e l'orientamento della fotocamera

2.2.4 Creare un'anteprima

L'elemento portante di un'applicazione basata sulla visione è un'anteprima in tempo reale di quello che la fotocamera sta vedendo. L'API Camera non contiene una classe apposita che rappresenti l'anteprima, sarà lo sviluppatore a doverla realizzare.

La classe in questione dovrà implementare l'interfaccia `SurfaceHolder.Callback`, utilizzata per trasferire i dati dalla fotocamera hardware all'applicazione e mantenere un riferimento alla Camera della quale mostrerà l'anteprima.

```
1 class CameraPreview(context: Context, private val surfaceView: SurfaceView) : ViewGroup(
    context), SurfaceHolder.Callback {
2
3     private val holder: SurfaceHolder = surfaceView.holder
4     init {
5         addView(surfaceView)
6         holder.addCallback(this)
7     }
8
9     // Metodo per aggiornare il riferimento alla Camera e avviare l'anteprima
10    fun setCamera(camera: Camera?) {
11        // Verifica se la Camera impostata e' diversa da quella attuale
12        if (mCamera == camera) {
13            return
14        }
15        stopPreviewAndFreeCamera()
16        // Aggiorna il riferimento
17        mCamera = camera
18        // Recupera le dimensioni di anteprima supportate e imposta l'anteprima
19        mCamera?.apply {
20            mSupportedPreviewSizes = parameters.supportedPreviewSizes
21            requestLayout()
22            try {
23                setPreviewDisplay(holder)
24            } catch (e: IOException) {
25                e.printStackTrace()
```



```

26         }
27         // Il metodo startPreview deve essere richiamato prima di poter fare foto
28         startPreview()
29     }
30 }
31
32 // Metodo per fermare l'anteprima e rilasciare la Camera
33 private fun stopPreviewAndFreeCamera() {
34     mCamera?.apply {
35         stopPreview()
36         mCamera?.also { camera ->
37             camera.release()
38             mCamera = null
39         }
40     }
41 }

```

I metodi di callback **surfaceChanged** e **surfaceDestroyed** possono per esempio essere utilizzati per aggiornare le dimensioni dell'anteprima e per chiuderla.

```

1 class CameraPreview(context: Context, private val surfaceView: SurfaceView) : ViewGroup(
    context), SurfaceHolder.Callback {
2
3     ...
4
5     override fun surfaceChanged(holder: SurfaceHolder, format: Int, width: Int, height:
        Int) {
6         mCamera?.apply {
7             parameters?.also { params ->
8                 params.setPreviewSize(previewSize.width, previewSize.height)
9                 requestLayout()
10                parameters = params
11            }
12            startPreview()
13        }
14    }
15
16    override fun surfaceDestroyed(holder: SurfaceHolder) {
17        mCamera?.stopPreview()
18    }
19
20    ...
21 }

```

È possibile implementare anche il **Camera.PreviewCallback** per manipolare i frame dell'anteprima in tempo reale.

```

1 class CameraPreview(context: Context, private val surfaceView: SurfaceView) : ViewGroup(
    context), SurfaceHolder.Callback {
2
3     ...
4

```

```

5     fun setCamera(camera: Camera?) {
6
7         ...
8
9         // Recupera le dimensioni di anteprima supportate e imposta l'anteprima
10        mCamera?.apply {
11
12            ...
13
14            camera.setPreviewCallback { data, camera ->
15                processPreviewFrame(data, camera)
16            }
17
18            ...
19
20        }
21    }
22
23    ...
24
25    private fun processPreviewFrame(data: ByteArray, camera: Camera) {
26        // Processa il frame dell'anteprima qui
27    }
28 }

```

2.2.5 Scattare una foto

Una volta che l'anteprima è stata avviata con il metodo **startPreview** possiamo chiamare il metodo **Camera.takePicture(shutter: Camera.ShutterCallback!, raw: Camera.PictureCallback!, jpeg: Camera.PictureCallback!)**.

Implementando il **Camera.PictureCallback** è possibile manipolare i dati ricevuti dalla fotocamera.

```

1 private lateinit var camera: Camera
2
3 private fun takePhoto() {
4     camera.takePicture(null, null, this)
5 }
6
7 override fun onPictureTaken(data: ByteArray?, camera: Camera?) {
8     // Salva l'immagine su file
9     try {
10         val filename = "photo.jpg"
11         val fileOutputStream = openFileOutput(filename, Context.MODE_PRIVATE)
12         // Percorso completo del file nell'archiviazione interna
13         val filePath = File(filesDir, filename).absolutePath
14         fileOutputStream.write(data)
15         fileOutputStream.close()
16     } catch (e: IOException) {
17         e.printStackTrace()
18     }
19 }

```

```

18     }
19     // Ripristina l'anteprima della fotocamera dopo lo scatto
20     camera?.startPreview()
21 }

```

Note

- È necessario ripristinare l'anteprima ogni volta che viene chiamato il metodo `Camera.takePicture(...)`

2.2.6 Registrare un video

Per poter registrare un video è necessario appoggiarsi alla classe **MediaRecorder** e aver richiesto i permessi per registrare l'audio (vedi sopra).

```

1 private var isRecording = false
2 private lateinit var camera: Camera
3
4 private fun startRecording() {
5     mediaRecorder = MediaRecorder().apply {
6         setCamera(camera)
7         setAudioSource(MediaRecorder.AudioSource.CAMCORDER)
8         setVideoSource(MediaRecorder.VideoSource.CAMERA)
9         setProfile(CamcorderProfile.get(CamcorderProfile.QUALITY_HIGH))
10        setOutputFile("video.mp4")
11        setPreviewDisplay(surfaceHolder.surface)
12        prepare()
13        start()
14    }
15    isRecording = true
16 }
17
18 private fun stopRecording() {
19     mediaRecorder.apply {
20         stop()
21         reset()
22         release()
23     }
24     isRecording = false
25 }
26 }

```

2.2.7 Impostare i Camera Parameters

In base alle funzionalità disponibili a livello hardware è possibile sfruttare diversi parametri della fotocamera. Una lista dei `Camera.Parameters` con una descrizione può essere trovata **qui**.

```

1 val params = camera.parameters
2
3 // Impostare la modalita' di scatto

```

```

4 params.cameraMode = Camera.Parameters.CAMERA_MODE_CONTINUOUS_SHOT
5 camera.parameters = params
6
7 // Impostare la messa a fuoco automatica
8 params.focusMode = Camera.Parameters.FOCUS_MODE_AUTO
9 camera.parameters = params
10
11 // Impostare l'esposizione
12 params.exposureCompensation = 1 // Imposta il valore dell'esposizione (es. +1 per
    aumentare l'esposizione, -1 per diminuirla)
13 camera.parameters = params
14
15 // Impostare il bilanciamento del bianco
16 params.whiteBalance = Camera.Parameters.WHITE_BALANCE_AUTO
17 camera.parameters = params
18
19 // Impostare la modalita' flash
20 params.flashMode = Camera.Parameters.FLASH_MODE_AUTO
21 camera.parameters = params
22
23 // Gestire lo zoom
24 if (params.isZoomSupported) {
25     val maxZoom = params.maxZoom // Ottieni il livello di zoom massimo supportato
26     params.zoom = maxZoom / 2 // Imposta il livello di zoom a meta' del valore massimo
27     camera.parameters = params
28 }

```

2.3 Pro e Contro

Camera è una libreria Android introdotta da Google per semplificare lo sviluppo delle funzionalità della fotocamera nelle applicazioni Android. Ecco alcuni pro e contro di Camera:

2.3.1 Pro

- Controllo diretto dell'hardware: Camera offre un controllo diretto sull'hardware della fotocamera del dispositivo. Ciò consente di accedere a funzionalità specifiche della fotocamera come il controllo del flash, la messa a fuoco manuale e l'esposizione.
- Facilità d'uso: Camera offre un'interfaccia relativamente semplice e diretta per l'accesso alla fotocamera. È possibile acquisire l'anteprima, scattare foto e registrare video utilizzando un insieme di metodi e callback ben definiti.
- Accesso ai frame dell'anteprima: Camera consente di accedere in tempo reale ai frame dell'anteprima della fotocamera, consentendo agli sviluppatori di visualizzare e elaborare i frame prima di catturare un'immagine o registrare un video.

2.3.2 Contro

- Complessità della gestione: Camera fornisce un'interfaccia di basso livello (vicina all'hardware) e richiede che sia lo sviluppatore ad occuparsi personalmente di gestire il ciclo di vita della fotocamera e dell'implementazione dell'anteprima .
- Limitazioni hardware: le prestazioni dell'API Camera dipendono molto dall'hardware del dispositivo. Questo potrebbe influire sulla qualità dell'immagine e sulla velocità di cattura.
- Deprecata: Camera è stata deprecata a partire da *Android 5.0 (API level 21)* e superiori in favore di Camera2 e CameraX e potrebbe non ricevere ulteriori aggiornamenti.

2.3.3 Conclusioni

Camera è stata la prima classe finalizzata al controllo della fotocamera dei dispositivi Android. Essendo deprecata da Android 5.0, essa non riceve più aggiornamenti, a differenza di quanto accade con Camera2 e CameraX, come vedremo nei capitoli successivi.

Camera consente di gestire gli use cases più comuni, come la visualizzazione di un'anteprima, la registrazione di video e la cattura di foto, e la modifica di alcuni parametri della fotocamera come l'esposizione e il flash. La lista da cui attingere per personalizzare i `Camera.Parameters` è tuttavia molto scarna se confrontata con quella dei parametri di Camera2, che avranno una sezione dedicata nel prossimo capitolo.

3 Classe Camera2

Il package `android.hardware.camera2` sostituisce la classe `Camera`, ormai deprecata. Esso consente di accedere alla fotocamera dei dispositivi android fornendone un'interfaccia che ne rende più completo il controllo e l'accesso a funzionalità avanzate.

Il funzionamento di una fotocamera gestita da questo pacchetto è riconducibile a quello di una pipeline: essa riceve le richieste per catturare un'immagine, la elabora utilizzando filtri, bilanciamento dei colori o altre operazioni di post-produzione e infine mostra l'immagine a schermo o semplicemente la salva sul dispositivo. Le richieste di accesso alle funzioni della camera vengono soddisfatte in base all'ordine di arrivo e più richieste possono essere gestite contemporaneamente.

3.1 Compatibilità

Camera2 è stata introdotta in *Android 5.0 (API level 21)* nel novembre 2014, sostituendo l'API1 con l'API2. Alcuni dispositivi con versione di android pari o superiore alla 5.0 potrebbero non supportare tutte le features di API2. Esistono infatti cinque diversi livelli di dispositivi, ottenibili dalla property `android.info.supportedHardwareLevel`, che supportano uno specifico insieme di features:

- **LEGACY**: dispositivi con capacità molto limitate, approssimativamente quelle di API1; API2 che opera in modalità retrocompatibile per i vecchi device android
- **LIMITED**: dispositivi che utilizzano solo alcune funzionalità di API2
- **FULL**: dispositivi che ampliano le funzionalità del livello LIMITED, aggiungendo il controllo manuale del flash o impostazioni post-processing
- **LEVEL_3**: dispositivi che permettono di avere funzionalità aggiuntive rispetto al livello FULL, come la registrazione di video e la cattura di immagini ad alta qualità (es. supporto per il riprocessamento YUV), immagini RAW e la configurazione dell'output
- **EXTERNAL**: dispositivi che forniscono una camera esterna (es. webcam USB) con funzionalità simili al livello LIMITED ma con possibili limitazioni aggiuntive, come ad esempio alcuni sensori o alcune informazioni sugli obiettivi della fotocamera

È possibile consultare la lista di tutte le capabilities mediante la property `android.request.availableCapabilities`. Alcune features non appartengono ad alcun livello hardware specifico e vengono identificate separatamente da `android.request.availableCapabilities`, come ad esempio il face detection (`android.statistics.info.availableFaceDetectModes`).

3.2 Funzionalità

3.2.1 CameraCaptureSession e CaptureRequest

Il package `camera2` introduce la possibilità di utilizzare più di uno stream proveniente da una singola fotocamera alla volta, ciascuno specializzato per uno specifico utilizzo, come la visualizzazione di

un'anteprima o la cattura di una foto. Ogni fotocamera, individuata come `CameraDevice`, può creare un'istanza della classe `CameraCaptureSession` per poter gestire gli stream. Un'istanza di `CameraDevice` deve ricevere una configurazione per ogni RAW frame, fornita da `CameraCaptureSession`, che specifica attributi della fotocamera come l'autofocus, l'esposizione o l'apertura. A causa di limitazioni hardware, in uno specifico momento può esserci una sola configurazione nel sensore della camera; questa configurazione prende il nome di active configuration. Una `CameraCaptureSession` descrive tutte le possibili pipeline collegate a un `CameraDevice`. Una volta creata una sessione, non è possibile aggiungere o eliminare pipeline. La `CameraCaptureSession` mantiene una coda di istanze di `CaptureRequest`; ciascuna istanza aggiunge in coda una configurazione e seleziona una o più pipeline disponibili per ricevere un frame dal `CameraDevice`. È possibile inviare più richieste durante una sessione di cattura ed ogni richiesta può cambiare la configurazione attiva e l'insieme delle pipeline.

3.2.2 HDR video

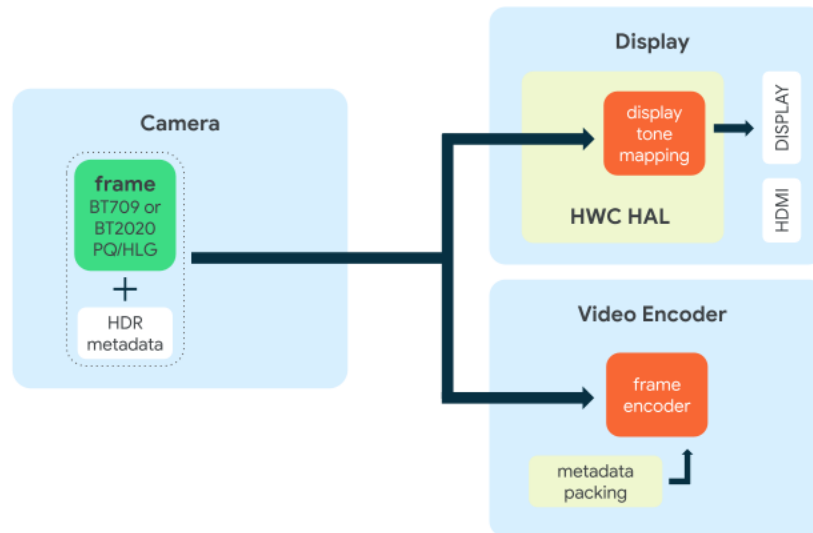
L'API2 fornisce la possibilità di visualizzare una preview e di registrare un video in HDR (High Dynamic Range) utilizzando la fotocamera del proprio dispositivo.

Questa funzionalità è stata annunciata nella presentazione "What's new in Android" del 2022 tra le principali novità introdotte da *Android 13 (API level 33)*, e ha come prerequisito il supporto del formato HLG10, ovvero richiede che il sensore della fotocamera supporti un output di 10 bit.

HDR è una tecnologia che consente di visualizzare un'ampia gamma di colori e di livelli di luminosità, in modo da ottenere immagini maggiormente fedeli e naturali, in particolar modo in ambienti molto luminosi o molto scuri, come una finestra illuminata dal sole o l'interno di una grotta. HDR migliora sensibilmente le prestazioni di SDR (Standard Dynamic Range), passando da un range di luminosità compreso tra 0.5 e 100 cd/mq (candele al metro quadro) a 0.05-1000 cd/mq.

Al momento della cattura di un frame, il framework Camera2 alloca un buffer per memorizzare l'output del sensore della fotocamera più eventuali metadati quali, ad esempio, il livello di esposizione. Dopo essere stato riempito, il buffer viene accodato per essere inviato alla superficie di output specificata da `CaptureRequest`, come un display o un codificatore video.





HDR Capture



3.2.3 Multi-camera API

Dall'avvento di *Android 9.0 Pie (API level 28)* nel 2018, è possibile l'utilizzo di più fotocamere in contemporanea sui dispositivi android compatibili. Al contrario, prima del rilascio di *Android P*, era possibile accedere soltanto ad uno dei sensori fisici della fotocamera.

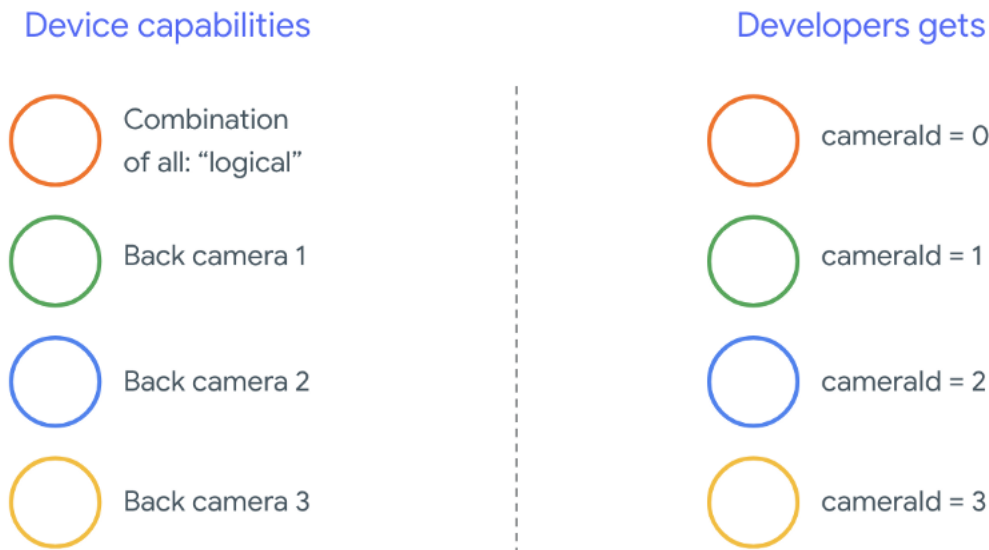
Device capabilities

-  Combination of all: "logical"
-  Back camera 1
-  Back camera 2
-  Back camera 3

Developers gets

-  cameraId = 1

Come anticipato, *Android 9.0* consente di accedere alle diverse fotocamere disponibili del dispositivo, come la grandangolare o la telescopica. In aggiunta alle fotocamere fisiche, l'API Multi-camera fornisce l'accesso alla fotocamera logica, un'astrazione software di tutti i sensori fisici che permette di utilizzare simultaneamente stream provenienti da due o più fotocamere fisiche del dispositivo attraverso un'unica interfaccia.



3.2.4 Controllo manuale dei parametri

API2 permette il controllo di una vasta gamma di parametri della fotocamera, in modo che l'utente possa modificarli manualmente.

Inizialmente l'utente dovrà creare un oggetto di tipo `CaptureRequest` mediante l'istruzione:

```
1 val captureRequestBuilder = cameraDevice.createCaptureRequest(CameraDevice.  
    TEMPLATE_PREVIEW)
```

`TEMPLATE_PREVIEW` è una costante che identifica il template predefinito fornito da API2 per richiedere la visualizzazione della preview della fotocamera selezionata. Tra gli altri template presenti nel package di `camera2` troviamo `TEMPLATE_STILL_CAPTURE` per la cattura di immagini, `TEMPLATE_RECORD` per la registrazione video, `TEMPLATE_VIDEO_SNAPSHOT` per la cattura di immagini durante la registrazione di un video.

Successivamente si dovrà chiamare il metodo `set(key: CaptureRequest.Key<T>!, value: T?)` per selezionare il parametro da modificare.

Ecco una lista dei principali parametri che `camera2` permette di modificare:

- **Bilanciamento del bianco:** è una tecnica che permette di ottenere una colorazione delle immagini più simile alla realtà, a seconda della temperatura del colore della luce ambientale, na-

turale o artificiale. Era già possibile modificare i valori di bilanciamento utilizzando la classe `Camera.Parameters`, come accennato nella sezione **2.2.7**.

```
1 // imposta il bilanciamento del bianco automatico
2 captureRequestBuilder.set(CaptureRequest.CONTROL_AWB_MODE, CaptureRequest.
    CONTROL_AWB_MODE_AUTO)
3
4 // in caso di luce diurna
5 captureRequestBuilder.set(CaptureRequest.CONTROL_AWB_MODE, CaptureRequest.
    CONTROL_AWB_MODE_DAYLIGHT)
6
7 // in ambienti illuminati da luci calde e di colore giallo-arancione come candele o
    lampadine a incandescenza
8 captureRequestBuilder.set(CaptureRequest.CONTROL_AWB_MODE, CaptureRequest.
    CONTROL_AWB_MODE_INCANDESCENT)
```

Questi parametri differiscono da quelli di `Camera.Parameters` per il nome delle costanti (ad esempio `WHITE_BALANCE_AUTO` invece di `CONTROL_AWB_MODE_AUTO` per il bilanciamento automatico) e il tipo (i `Camera.Parameters` sono di tipo `String`, mentre le properties di `CaptureRequest` sono `Int`). Si può inoltre bloccare il bilanciamento del bianco usando gli ultimi valori calcolati per le successive catture, trascurando eventuali cambi di illuminazione.

```
1 // la fotocamera terra' le stesse impostazioni di bilanciamento (determinate dagli
    ultimi valori calcolati) fino a quando il valore del parametro non verra'
    impostato a false
2 captureRequestBuilder.set(CaptureRequest.CONTROL_AWB_LOCK, true)
```

Qualora si desiderasse applicare il bilanciamento soltanto in una zona limitata dell'immagine, `CaptureRequest` fornisce il parametro `CONTROL_AWB_REGIONS`.

```
1 // passo come argomento la zona selezionata, ovvero un oggetto di tipo Array<
    MeteringRectangle!>
2 captureRequestBuilder.set(CaptureRequest.CONTROL_AWB_REGIONS, arrayOf(
    meteringRectangle))
```

- **Esposizione:** è la quantità di luce catturata dal sensore della fotocamera al momento dello scatto.

```
1 // imposta l'esposizione automatica
2 captureRequestBuilder.set(CaptureRequest.CONTROL_AE_MODE, CaptureRequest.
    CONTROL_AE_MODE_ON)
3
4 // imposta l'esposizione automatica con il flash sempre attivo
5 captureRequestBuilder.set(CaptureRequest.CONTROL_AE_MODE, CaptureRequest.
    CONTROL_AE_MODE_ON_ALWAYS_FLASH)
6
7 // imposta l'esposizione automatica con l'attivazione automatica del flash quando
    necessario
8 captureRequestBuilder.set(CaptureRequest.CONTROL_AE_MODE, CaptureRequest.
    CONTROL_AE_MODE_ON_AUTO_FLASH)
```

Inoltre è possibile impostare la compensazione dell'esposizione, ovvero l'aumento o la diminuzione dell'esposizione rispetto alla quantità di luce misurata automaticamente dalla fotocamera. Il range massimo di compensazione e gli step minimi per far variare l'Exposure Value (EV) vengono determinati rispettivamente da `android.control.aeCompensationRange` e `android.control.aeCompensationStep`.

```
1 // aumenta l'esposizione di 2 EV
2 captureRequestBuilder.set(CaptureRequest.CONTROL_AE_EXPOSURE_COMPENSATION, 6)
```

- **Immagini JPEG**

```
1 // ruota l'immagine JPEG catturata di 90 gradi
2 captureRequestBuilder.set(CaptureRequest.JPEG_ORIENTATION, JPEG_ORIENTATION_90)
3
4 // imposta la qualita' desiderata (un valore tra 1 e 100) per l'immagine,
  solitamente 85-95
5 captureRequestBuilder.set(CaptureRequest.JPEG_QUALITY, 90)
6
7 // e' possibile modificare anche la qualita' dell'immagine visualizzata in
  anteprima
8 captureRequestBuilder.set(CaptureRequest.JPEG_THUMBNAIL_QUALITY, 80)
9
10 // fornisce le coordinate GPS all'immagine catturata attraverso un oggetto di tipo
    Location
11 captureRequestBuilder.set(CaptureRequest.JPEG_GPS_LOCATION, location)
```

- **Hot Pixel:** gli hot pixel sono dei punti dell'immagine catturata in ambienti poco illuminati erroneamente più luminosi rispetto a quelli circostanti, solitamente dovuti a tempi di esposizione lunghi (sull'ordine dei secondi).

```
1 // correzione rapida degli hot pixel
2 captureRequestBuilder.set(CaptureRequest.HOT_PIXEL_MODE, HOT_PIXEL_MODE_FAST)
3
4 // correzione ad alta qualita' degli hot pixel; richiede un costo maggiore di
  elaborazione
5 captureRequestBuilder.set(CaptureRequest.HOT_PIXEL_MODE,
    HOT_PIXEL_MODE_HIGH_QUALITY)
```

- **Zero Shutter Lag:** la modalità Zero-Shutter-Lag (ZSL) riduce il ritardo della cattura effettiva dell'immagine dopo aver premuto il tasto dello scatto. Il parametro `CONTROL_ENABLE_ZSL` è stato aggiunto in *API level 26* e dev'essere impostato a `true` per attivare la modalità ZSL.

```
1 // attiva la modalita' ZSL
2 captureRequestBuilder.set(CaptureRequest.CONTROL_ENABLE_ZSL, true)
```

- **Auto-framing:** vengono applicate una serie di tecniche in modo automatico, ovvero il crop (ritaglio), il panning (permette di seguire il movimento del soggetto) e lo zoom, con lo scopo di enfatizzare il soggetto raffigurato. L'auto-framing sarà attivabile/disattivabile dall'avvento di

API level 34 di Android 14 (al momento della stesura di questo report, giugno 2023, non ancora rilasciato) tramite la property `CONTROL_AUTOFRAMING`.

```
1 // attiva l'auto-framing
2 captureRequestBuilder.set(CaptureRequest.CONTROL_AUTOFRAMING,
    CONTROL_AUTOFRAMING_ON)
```

- **Extended Scene Mode:** *API level 30* introduce il parametro `CONTROL_EXTENDED_SCENE_MODE` per gestire le modalità estese, in particolare la modalità bokeh.

```
1 // modalita' bokeh per fotografie
2 // 1 corrisponde a CONTROL_EXTENDED_SCENE_MODE_BOKEH_STILL_CAPTURE
3 captureRequestBuilder.set(CaptureRequest.CONTROL_EXTENDED_SCENE_MODE, 1)
4
5 // modalita' bokeh per video
6 // 2 corrisponde a CONTROL_EXTENDED_SCENE_MODE_BOKEH_CONTINUOUS
7 captureRequestBuilder.set(CaptureRequest.CONTROL_EXTENDED_SCENE_MODE, 2)
8
9 // modalita' estesa disattivata
10 // 0 corrisponde a CONTROL_EXTENDED_SCENE_MODE_DISABLED
11 captureRequestBuilder.set(CaptureRequest.CONTROL_EXTENDED_SCENE_MODE, 0)
```

Con Android 14 sarà possibile modificare manualmente il valore dell'effetto selezionato. Ad esempio, in modalità bokeh si potrà controllare il livello di sfocatura e in modalità night la luminosità dell'immagine catturata.

```
1 // il valore di strength deve assumere un valore tra 0 e 100
2 captureRequestBuilder.set(CaptureRequest.EXTENSION_STRENGTH, strength)
```

- **Zoom ratio:** precedentemente all'avvento di *API level 30* era necessario utilizzare `android.scaler.cropRegion` per controllare il livello dello zoom. Con l'introduzione di questa API, corrispondente ad *Android 11*, è possibile sfruttare la key `CONTROL_ZOOM_RATIO` per selezionare il fattore di zoom desiderato in un range di possibili valori definiti in `android.control.zoomRatioRange`.

```
1 // zoomRatio deve essere di tipo Float
2 captureRequestBuilder.set(CaptureRequest.CONTROL_ZOOM_RATIO, zoomRatio)
```

- **Correzione delle distorsioni:** da *API level 28* in poi è possibile migliorare l'accuratezza dell'immagine catturata cercando di limitare le distorsioni geometriche dovute alle caratteristiche intrinseche delle lenti della fotocamera.

```
1 // correzione della distorsione impostata su HIGH_QUALITY (=2), puo' causare un
    calo del frame rate
2 // in alternativa puo' essere impostato il valore FAST (=1) che non abbassa il
    frame rate una volta applicata la correzione o ad OFF (=0) se non si vuole
    correggere la distorsione
3 captureRequestBuilder.set(CaptureRequest.DISTORTION_CORRECTION_MODE, 2)
```

- **Apertura del diaframma:** identifica l'ampiezza della sezione dell'obiettivo attraverso il quale può passare la luce che va a colpire il sensore della fotocamera. Camera2 fornisce la chiave `LENS_APERTURE` per modificare questo valore conosciuto anche come rapporto focale, o f-stop, ovvero il rapporto tra la lunghezza focale dell'obiettivo e il diametro del diaframma. I possibili valori (f/stop) appartengono ad una progressione geometrica (f/1, f/1.4, f/2, f/2.8, ...) ed è importante sottolineare che a valori inferiori corrisponde un'apertura maggiore e quindi ad una luminosità dell'obiettivo più alta.

```
1 // apertura del diaframma pari a f/2.8
2 captureRequestBuilder.set(CaptureRequest.LENS_APERTURE, 2.8f)
3
4 // lunghezza focale pari a 35mm, obiettivo grandangolare
5 captureRequestBuilder.set(CaptureRequest.LENS_FOCAL_LENGTH, 35f)
```

- **Face detection:** in `CaptureRequest` è presente la property `STATISTICS_FACE_DETECT_MODE` per attivare il riconoscimento facciale.

```
1 // SIMPLE: algoritmi di riconoscimento semplici
2 captureRequestBuilder.set(CaptureRequest.STATISTICS_FACE_DETECT_MODE, 1)
3
4 // FULL: restituisce tutti i metadati del volto, utile per autenticazione con face
  detection
5 captureRequestBuilder.set(CaptureRequest.STATISTICS_FACE_DETECT_MODE, 2)
```

- **Immagini RAW:** il builder per le capture request `CaptureRequest.Builder`, tra le fila dei suoi metodi pubblici, presenta `addTarget(outputTarget: Surface)` che aggiunge la surface indicata alla lista dei possibili output per la richiesta in questione. Come anticipato, con camera2 possono essere aggiunte più surface per una singola richiesta. Ecco un esempio di cattura di immagini RAW, ottenute senza elaborare i dati acquisiti dal sensore della fotocamera:

```
1 // ottiene un'istanza di ImageReader specificando il formato RAW
2 val imageReader = ImageReader.newInstance(width, height, ImageFormat.RAW_SENSOR,
  maxImages)
3
4 // crea la richiesta per la cattura di immagini RAW
5 val captureRequestBuilder = cameraDevice.createCaptureRequest(CameraDevice.
  TEMPLATE_STILL_CAPTURE).apply {
6   addTarget(previewSurface)
7   addTarget(imageReader.surface)
8   // CONTROL_CAPTURE_INTENT specifica il caso d'uso dell'immagine acquisita,
9   set(CaptureRequest.CONTROL_CAPTURE_INTENT, 2) // 2 = STILL_CAPTURE
10 }
```

Dopo aver modificato tutti i parametri di interesse, si aggiorna la `CameraCaptureSession`.

```
1 val captureRequest = captureRequestBuilder.build()
2 cameraCaptureSession.setRepeatingRequest(captureRequest, null, null)
```

3.3 Pro e Contro

3.3.1 Pro

- Multi-camera API: la possibilità di accedere alla fotocamera logica permette la gestione di più di una fotocamera fisica simultaneamente attraverso un'unica interfaccia.
- Multi-stream: un'app che utilizza il package `camera2` può sfruttare molteplici stream di frame per diversi scopi. Ad esempio, un lettore di codici a barre utilizza uno stream per la preview e un altro per la lettura del codice a barre.
- Parametri della fotocamera: si possono modificare manualmente alcuni parametri della fotocamera del dispositivo per migliorare la qualità delle foto, come l'apertura del diaframma, il controllo dello zoom, o usare modalità avanzate come Bokeh e Night.
- Acquisizione RAW: è la cattura diretta di immagini dal sensore della fotocamera senza alcuna elaborazione. Ciò è fattibile se tra i possibili valori del campo `REQUEST_AVAILABLE_CAPABILITIES` di `CameraCharacteristics` c'è `REQUEST_AVAILABLE_CAPABILITIES_RAW`.
- Video HDR: *camera2* consente di visualizzare un'anteprima e registrare video in HDR (High Dynamic Range) usando la fotocamera del proprio dispositivo per garantire una gamma più ampia di livelli di luminosità.
- Burst capture: viene introdotta la possibilità di scattare una serie di foto in rapida successione grazie al metodo `setRepeatingBurst` di `CameraCaptureSession`.
- Nuove versioni: il pacchetto riceve continui aggiornamenti; le API successive al rilascio di *Android 5.0* hanno portato novità consistenti a *camera2* come la gestione avanzata dello zoom o l'aggiunta di parametri per l'ottenimento di foto e video più accurati.

3.3.2 Contro

- Capacità delle pipeline: la gestione dei flussi di elaborazione può essere ostacolata dalla struttura a pipeline di *camera2*. Infatti un elemento della pipeline potrebbe non essere in grado di gestire l'imminente flusso di dati e quindi causare un drop dei frame.
- Costi di memoria: l'elaborazione dei frame richiede un costo di prestazioni non trascurabile, in particolare il costo di memoria aumenta linearmente.
- Compatibilità: il package *camera2* è stato introdotto in *API level 21*, perciò non è compatibile con i dispositivi che utilizzano versioni precedenti ad *Android 5.0*. Per di più alcune funzionalità sono state aggiunte in API di più recente rilascio, ad esempio i video in HDR hanno esordito in *Android 13* ed è quindi più comune che utenti non possano usufruire di queste funzionalità, nonostante la maggior parte del pacchetto sia da loro utilizzabile.

3.3.3 Conclusioni

Riassumendo, il package *camera2* ha l'obiettivo di prendere le funzionalità base della classe *Camera*, ovvero la visualizzazione di anteprime, scattare foto e registrare video, e ampliarle per gestire altri use cases come la burst capture o l'acquisizione di immagini RAW.

Un punto cardine di questo pacchetto è certamente l'uso contemporaneo di più fotocamere di un dispositivo, ottenibili come istanze di *CameraDevice*, che vengono gestite dalla classe *CameraManager* utilizzando i suoi metodi fondamentali come *getCameraIdList* e *openCamera*, usufruendo della camera logica.

Un altro argomento trattato in questo capitolo è quello delle richieste di cattura personalizzate, modificando i valori delle key di *CaptureRequest* per ottenere immagini di qualità migliore a seconda della luce ambientale, per eliminare gli hot pixel o le aberrazioni geometriche, o per attivare dei filtri.

I continui aggiornamenti a *camera2* mantengono in vita il pacchetto che, come visto nella sezione precedente, aggiunge nuovi parametri a *CaptureRequest* per gestire più approfonditamente diversi casi d'uso come l'effetto bokeh o lo zoom.

4 Classe CameraX

CameraX è una libreria appartenente alla suite di librerie *Android Jetpack*, che facilita lo sviluppo di applicazioni che implementano l'utilizzo della fotocamera, semplificando lo sviluppo di codice, rendendolo funzionante sulla maggior parte delle versioni e dispositivi Android, ed è compatibile con i dispositivi che montano Android 5.0 (livello API 21) o superiori, permettendone l'utilizzo al 98% dei dispositivi Android esistenti.

4.1 Introduzione

4.1.1 Versioni

L'inizio dello sviluppo e primo rilascio di una versione di CameraX è stato nell'agosto 2019, con la versione **Camera-Extensions and Camera-View Version 1.0.0-alpha01**, con la seguente nota di rilascio: "New library for future Camera Extensions for accessing effects on supported devices. This library is a work in progress".

La prima versione stabile, **Version 1.0.0** è stata resa disponibile nel maggio 2021, mentre, alla data di stesura di questo capitolo, l'ultima versione stabile disponibile è la **1.2.3**, rilasciata il 24 maggio 2023; lo stesso giorno è stata rilasciata anche la versione **1.3.0-alpha07**.

4.1.2 Creazione

In occasione di Google I/O, ossia una conferenza annuale organizzata da Google per gli sviluppatori ("I/O" sta appunto per "Input/Output"), vengono presentati nuovi prodotti, nuove tecnologie, piattaforme e importanti aggiornamenti di prodotti, come nuove versioni di Android e nuove funzionalità. Nell'evento del 2019, Google ha presentato CameraX, una nuova libreria Jetpack, creata per facilitare lo sviluppo di applicazioni che utilizzano la fotocamera del dispositivo.

Gli obiettivi principali sono:

- Aumentare la facilità di utilizzo della camera, fornendo diversi casi d'uso come l'anteprima, acquisizione del immagini e altre funzionalità descritte successivamente.
- Garantire la coerenza tra i dispositivi, poiché i diversi marchi possono gestire l'integrazione hardware/software in modi differenti. Ciò può creare difficoltà per gli sviluppatori nell'utilizzo di HW, come la fotocamera.

Quindi CameraX permette di "livellare" le differenze, semplificando l'utilizzo della fotocamera, consentendo di scrivere un codice che funziona con un'ampia gamma di dispositivi e offrire un'ottima qualità fotografica anche sui modelli di smartphone entry-level.

In questo modo Google vuole far sì che gli sviluppatori riescano a offrire applicazioni in grado di fornire una qualità grafica paragonabile a quelle dell'app fotocamera native sui dispositivi. Indipendentemente dal tipo di smartphone, dall'entry-level al top di gamma, fornendo un'esperienza fotografica di qualità elevata a tutti gli utenti.

4.1.3 App di terze parti

Si può notare come in app di terzi, come app social o di messaggistica, la qualità della fotocamera non è comparabile con quella dell'app nativa. Il problema sta nel fatto che queste app non sono in grado di usufruire di tutte le caratteristiche software e hardware del dispositivo.

In termini più tecnici, manca un framework unificato per la fotografia. Per esempio, se uno smartphone offre più di una fotocamera, lo sviluppatore deve aggiungerle manualmente. In un quadro già molto complicato si aggiungono funzioni software esclusive, come la modalità ritratto o la Night Mode, e il fatto che ogni produttore opera sostanzialmente come meglio crede, senza un coordinamento preciso con tutti gli altri. Queste differenze causano difficoltà agli sviluppatori che devono adattare le loro app alle diverse implementazioni e alle funzioni specifiche di ciascun produttore.

4.1.4 Funzionamento e estensioni

Google, con CameraX, intende risolvere il problema: essa viene usata come **libreria "ponte"** collegandosi all'API Camera2, attualmente già presente nei dispositivi.

Sostanzialmente CameraX verifica quali funzionalità sono dichiarate dal produttore in Camera2 e le rende utilizzabili.

Un altro vantaggio è che permette di usare delle librerie aggiuntive, chiamate **Camera extensions**, che permettono a CameraX di usufruire funzionalità software avanzate, come:

- **Auto:** adatta automaticamente la modalità con cui scattare la foto, in base al contesto della scena. Ad esempio, in condizioni di scarsa illuminazione, può passare alla modalità Notte o per i ritratti può applicare il Face Retouch o l'effetto Bokeh.
- **Bokeh:** mette in evidenza il soggetto in primo piano; solitamente usata per scattare foto ritratto di persone, mettendolo in risalto rispetto allo sfondo, che viene sfocato.
- **Face Retouch:** applica dei filtri che permettono di ritoccare i difetti della pelle e del volto in generale.
- **High Dynamic Range (HDR):** consente di acquisire una gamma di colori più ricca. La tecnica consiste nell'effettuare scatti multipli allo stesso soggetto ma con diverse esposizioni e successivamente unirle in una sola foto; questo consente di non avere una sottoesposizione o sovraesposizione nell'immagine e quindi di avere un'esposizione corretta sia nelle aree chiare che in quelle scure.
- **Night:** schiarisce le foto in situazioni di scarsa illuminazione, senza l'utilizzo del flash. La fotocamera scatta diverse foto a vari valori di esposizione per poi unirle in una sola: questo consente di catturare più luce ottenendo una qualità notturna superiore ad un semplice scatto. Per usare questa funzione però è necessario che l'utente tenga fermo il telefono per qualche secondo mentre la fotocamera acquisisce i dati.



Figura 1: Comparazione tra immagine senza (sulla sinistra) e con (sulla destra) effetto Bokeh



Figura 2: Comparazione tra immagine senza (sulla sinistra) e con (sulla destra) effetto HDR



Figura 3: Comparazione tra immagine senza (sulla sinistra) e con (sulla destra) effetto Night

4.2 Architettura

4.2.1 Struttura

La libreria CameraX permette di interfacciarsi con la fotocamera di un dispositivo attraverso un'astrazione, chiamata **caso d'uso**, per semplificare la scrittura del codice e gestire la complessità associata all'accesso alla fotocamera, permettendo quindi agli sviluppatori di concentrarsi maggiormente sulle funzionalità piuttosto che sull'implementazione e l'accesso alla camera stessa.

Sono disponibili i seguenti casi d'uso:

- **Preview:** accetta un oggetto surface, come PreviewView. Mostra in tempo reale l'immagine proveniente dalla fotocamera sullo schermo del dispositivo. È utilizzata per fornire all'utente

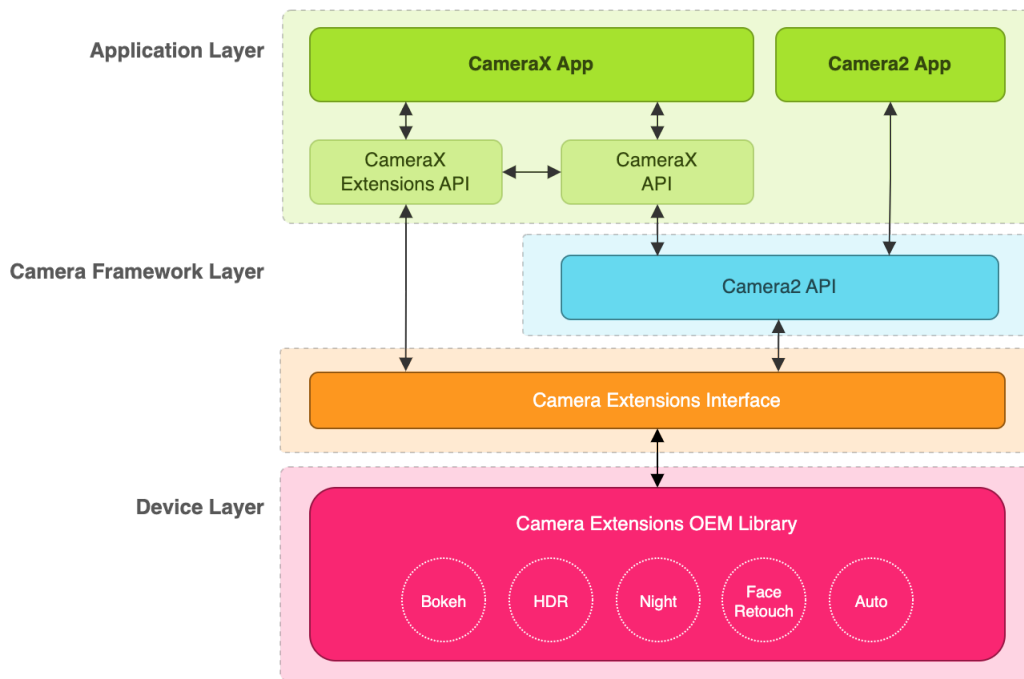


Figura 4: Architettura delle estensioni di CameraX

una visualizzazione dell'immagine che la fotocamera sta catturando, consentendo di inquadrare al meglio il soggetto.

- **Image Analysis:** fornisce buffer accessibili dalla CPU per l'analisi dell'immagine, ad esempio per le applicazioni di machine learning. Questo caso d'uso consente di ottenere l'accesso ai dati dell'immagine in tempo reale per eseguire elaborazioni o analisi personalizzate sulla fotocamera del dispositivo, ad esempio per il riconoscimento di oggetti, filtri o effetti in tempo reale.
- **Image Capture:** questo caso d'uso consente di scattare foto utilizzando la fotocamera del dispositivo e salvare l'immagine risultante su memoria interna o esterna. È il caso d'uso principale per l'acquisizione di immagini statiche tramite CameraX.
- **Video Capture:** consente di registrare audio e video utilizzando i microfoni e le fotocamere del dispositivo.

Questi casi d'uso forniscono funzionalità specifiche per l'interazione con la fotocamera del dispositivo utilizzando CameraX, semplificando lo sviluppo di applicazioni che coinvolgono la cattura di immagini o video, l'analisi delle immagini o la visualizzazione dell'anteprima in tempo reale.

4.2.2 API model

Per lavorare con la libreria, è necessario specificare le seguenti informazioni:

1. Quando viene selezionato il caso d'uso desiderato, quindi Preview, Image Analysis, Image Capture o Video Capture, si devono configurare le opzioni specifiche per ogni caso d'uso, come la risoluzione dell'immagine, il formato di output, le dimensioni della finestra di anteprima, ecc.
2. Dopo aver configurato il caso d'uso, è possibile specificare cosa fare con i dati di output generati dalla fotocamera. È possibile collegare listener ai casi d'uso in modo da poter elaborare, visualizzare o salvare i dati in base alle proprie esigenze.
3. Il flusso previsto per l'utilizzo di CameraX è strettamente collegato agli Android Architecture Lifecycles, come ad esempio l'attività o il frammento dell'applicazione. CameraX offre un supporto integrato per l'integrazione con questi lifecycles, consentendo di definire quando abilitare o disabilitare la fotocamera in base al ciclo di vita dell'applicazione.

I casi d'uso vengono configurati con i metodi `set()` e finalizzati con il metodo `build()`. Ogni oggetto fornisce un insieme di API specifiche per il caso d'uso. Ad esempio, il caso d'uso di acquisizione delle immagini fornisce una chiamata di metodo `takePicture()`.

Utilizzando CameraX non è necessario chiamare i metodi per interrompere l'utilizzo della fotocamera, quindi non è necessario l'utilizzo dei `onResume()` e `onPause()`, infatti viene creato un ciclo di vita a cui associare la telecamera tramite il comando `cameraProvider.bindToLifecycle()`. Il ciclo di vita informa CameraX quando configurare la sessione di acquisizione della fotocamera e garantisce che lo stato di quest'ultima cambi in modo appropriato per adattarsi alle transizioni del ciclo di vita.

```
1 private var imageCapture: ImageCapture? = null
2 lateinit var camera : androidx.camera.core.Camera
3 ...
4 // Creazione di imageCapture, per consentire di scattare foto
5 imageCapture = ImageCapture.Builder().build()
6
7 // lego imageCapture con il ciclo di vita della camera
8 camera = cameraProvider.bindToLifecycle(this, cameraSelector, preview, imageCapture)
```

4.2.3 Ciclo di vita

CameraX è legato ad un ciclo di vita per determinare quando aprire la fotocamera, quando creare una sessione di acquisizione, quando fermarsi e chiudersi. Le API dei casi d'uso forniscono chiamate di metodo e callback per monitorare i progressi.

È possibile associare alcuni mix di casi d'uso a un singolo ciclo di vita.

Quando l'applicazione deve supportare casi d'uso che non possono essere combinati, si può procedere in uno dei seguenti modi:

- Raggruppare i casi d'uso compatibili in più di un frammento e poi passare da un frammento all'altro.
- Creare un componente del ciclo di vita personalizzato e usarlo per controllare manualmente il ciclo di vita della telecamera.

Se si disaccoppiano i proprietari del ciclo di vita dei casi d'uso della vista e della fotocamera (ad esempio, se si utilizza un ciclo di vita personalizzato o un frammento di mantenimento), è necessario assicurarsi che tutti i casi d'uso siano svincolati da CameraX utilizzando `ProcessCameraProvider.unbindAll()` o svincolando ogni singolo caso d'uso. In alternativa, quando si legano i casi d'uso a un ciclo di vita, si può lasciare che CameraX gestisca l'apertura e la chiusura della sessione di acquisizione e lo svincolo dei casi d'uso.

Se le funzionalità della fotocamera sono strettamente legate al ciclo di vita di un singolo componente, come un'Activity o un Fragment, utilizzare il ciclo di vita di tale componente per gestire i casi d'uso desiderati garantisce che la telecamera sia pronta quando il componente è attivo e che venga liberata correttamente senza sprechi di risorse.

4.2.4 Permessi

Il permesso **principale** da richiedere per permettere l'utilizzo della camera è

```
1 <uses-permission android:name="android.permission.CAMERA"/>
```

Altri permessi utili sono `android.permission.RECORD_AUDIO` per consentire la registrazione dell'audio e `android.permission.WRITE_EXTERNAL_STORAGE` per salvare eventualmente i file in una memoria esterna

4.2.5 Dipendenze

Inoltre per l'utilizzo di CameraX è necessaria l'aggiunta delle seguenti dipendenze in `build.gradle` Module :app

```
1 def camerax_version = "1.3.0-alpha07"
2 implementation "androidx.camera:camera-core:${camerax_version}"
3 implementation "androidx.camera:camera-camera2:${camerax_version}"
4 implementation "androidx.camera:camera-lifecycle:${camerax_version}"
5 implementation "androidx.camera:camera-video:${camerax_version}"
6 implementation "androidx.camera:camera-view:${camerax_version}"
7 implementation "androidx.camera:camera-mlkit-vision:${camerax_version}"
8 implementation "androidx.camera:camera-extensions:${camerax_version}"
```

4.3 Configurazione

4.3.1 Panoramica

Ogni caso d'uso fornito da CameraX è configurabile. Questo consente di controllare vari aspetti dei casi d'uso.

Ad esempio, con il caso d'uso di acquisizione delle immagini, come il codice sotto mostra, è possibile impostare la modalità flash, in questo caso impostandola in Auto.

```
1 val imageCapture = ImageCapture.Builder().
2     setFlashMode(ImageCapture.FLASH_MODE_AUTO).
3     build()
```

Inoltre questi campi sono modificabili anche in seguito, esempio se voglio disattivare il flash mi basta inserire il seguente codice:

```
1 imageCapture.flashMode = ImageCapture.FLASH_MODE_OFF
```

Tutti i casi d'uso sono configurabili.

4.3.2 Configurazione automatica

CameraX offre automaticamente funzionalità adattate al dispositivo su cui viene eseguita l'applicazione. Ad esempio, se non viene specificata una risoluzione o la risoluzione specificata non è supportata, CameraX determinerà automaticamente la migliore risoluzione da utilizzare. Tutto ciò viene gestito dalla libreria, evitando di scrivere codice specifico per ogni dispositivo.

L'obiettivo principale di CameraX è avviare con successo una sessione della fotocamera. Ciò implica che CameraX faccia compromessi sulla risoluzione e sul rapporto di aspetto in base alle capacità del dispositivo.

Alcuni motivi per cui CameraX cambia automaticamente questi valori sono: il dispositivo non supporta la risoluzione richiesta, non supporta certi formati, o non li supporta su determinati aspect ratios.

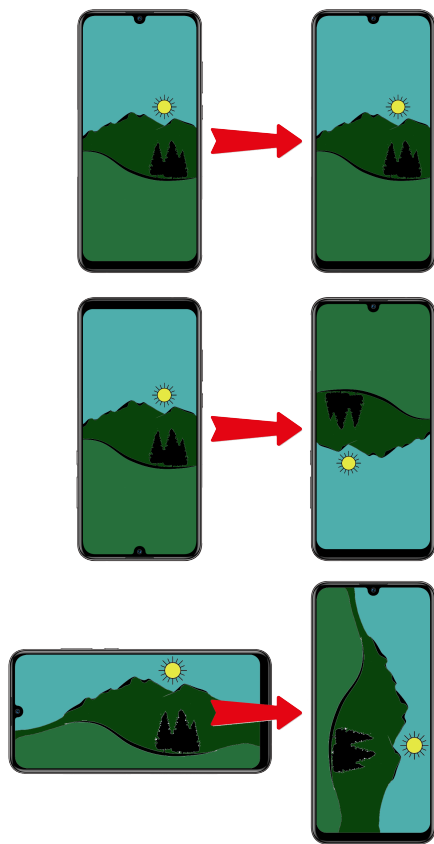
Il continuo di questo capitolo tratterà di come impostare le configurazioni più utili.

4.3.3 Rotazione

Come comportamento di default, l'orientamento della camera corrisponde a quello dello schermo quando viene creata la camera.

Per impostare l'orientamento si utilizza il comando `imageCapture.targetRotation = rotation`: questa impostazione è modificabile senza eliminare il ciclo di vita della camera. L'esempio seguente mostra come modificare l'orientamento della foto al variare dell'orientamento del telefono. Importante: non è possibile assegnare un qualsiasi valore alla rotazione, bensì quest'ultimo deve per forza essere multiplo di 90, ossia, deve corrispondere all'orientamento portrait, reverse portrait, landscape o reverse landscape.

```
1 val imageCapture = ImageCapture.Builder().build()
2
3 val orientationEventListener = object : OrientationEventListener(this as Context) {
4     override fun onOrientationChanged(orientation : Int) {
5         // Trasforma il valore da orientamento, che e' in gradi, in un valore
6         // impostabile in imageCapture.targetRotation
7         val rotation : Int = when (orientation) {
8             in 45..134 -> Surface.ROTATION_270
9             in 135..224 -> Surface.ROTATION_180
10            in 225..314 -> Surface.ROTATION_90
11            else -> Surface.ROTATION_0
12        }
13    }
14 }
```



Esempio: se la camera viene costruita con il telefono in portrait e viene scattata la foto, quest'ultima sarà orientata correttamente. Tuttavia, senza le dovute modifiche, se viene girato lo schermo in landscape o reverse portrait, l'immagine scattata sarà orientata in maniera errata rispetto all'orientamento del telefono. Inoltre, se la foto viene scattata in reverse portrait, verrà salvata capovolta.

Figura 5: A sinistra viene mostrato come viene scattata la foto, mentre a destra come quest'ultima viene visualizzata nella galleria

```

12
13     imageCapture.targetRotation = rotation
14 }
15 }
16 orientationEventListener.enable()

```

É possibile modificare anche l'orientamento del video nello stesso modo, usando `VideoCapture` al posto di un'istanza di `ImageCapture`.

4.3.4 Crop rect

É possibile ritagliare l'immagine/video che `CameraX` restituirà e la grandezza della `Preview`. Di default la grandezza dell'immagine visualizzata corrisponde al rettangolo del buffer. Il rettangolo di buffer o `buffer rect` è un concetto che viene utilizzato nella fotografia digitale e corrisponde all'area in cui la camera è in grado di recuperare i dati dall'ambiente; detta in un altro modo, corrisponde alla

dimensione del sensore della fotocamera che cattura l'immagine.

Questa grandezza è modificabile tramite l'utilizzo di `ViewPort` e `UseCaseGroup`.

4.3.5 Selezione Camera

In maniera automatica CameraX si occupa della selezione della migliore camera in base alle esigenze e al caso. É possibile scegliere manualmente la camera da utilizzare all'interno della costruzione di `ProcessCameraProvider`, classe fornita da CameraX API per gestire l'accesso alle funzionalità della fotocamera del dispositivo; esempio per eseguire il cambio camera:

```
1      cameraSelector =
2          if (condition)
3              CameraSelector.DEFAULT_BACK_CAMERA
4          else
5              CameraSelector.DEFAULT_FRONT_CAMERA
6      try {
7          cameraProvider.unbindAll()
8          camera = cameraProvider.bindToLifecycle(this, cameraSelector, preview,
9          imageCapture)
10         // in quato cambio la camera
11         cameraControl = camera.cameraControl
12     } catch(e: Exception) {
13         Log.e(TAG, "Build failed", e)
14     }
```

4.3.6 Multi Camera

Dalla versione 1.3 di CameraX è possibile anche l'utilizzo multiplo di più camere in contemporanea: ad esempio è possibile utilizzare simultaneamente sia la camera posteriore che anteriore per registrare video e scattare foto.

4.3.7 Risoluzione

É possibile modificare la risoluzione tramite la quale CameraX registra video e cattura foto. Se questa non viene impostata, CameraX lo fa automaticamente selezionando la risoluzione più adeguata.

4.3.8 Zoom

Una delle funzionalità più utili è il controllo dello Zoom. Questo viene fatto tramite `CameraControl` ed è possibile modificarlo in due modi:

- **setZoomRatio()**: consente di impostare il valore dello zoom specifico, questo valore va da `CameraInfo.getZoomState().getValue().getMinZoomRatio()`, ossi il valore minimo dello zoom effettuabile da quella camera a `CameraInfo.getZoomState().getValue().getMaxZoomRatio()`, quindi il valore massimo dello zoom effettuabile da quella camera.

- **setLinearRatio()**: tramite questo metodo invece è possibile impostare il valore dello zoom tra 0 e 1 compresi, dove 0 corrisponde allo zoom minimo di quella camera, e 1 allo zoom massimo.

É anche possibile recuperare il valore corrente di zoom tramite `ZoomState.getZoomRatio()` o `ZoomState.getLinearZoom()` dove `ZoomState` è un'istanza ottenuta da `CameraInfo.getZoomState()`.

4.3.9 Flash

Tramite `CameraX` è possibile anche il controllo del flash.

É possibile innanzitutto controllare se è presente l'unità hardware per il flash tramite il comando `CameraInfo.hasFlashUnit()`, successivamente è possibile impostarne lo stato tramite `CameraControl.enableTorch(boolean)` dove il valore booleano passato corrisponde ad acceso se è `true` o spento se è `false`. Inoltre `CameraX` si prende in carico il controllo del flash quando si scatta una foto.

In altre parole è possibile settare tramite il comando `imageCapture?.flashMode = ImageCapture.FLASH_MODE_XXX`, dove `FLASH_MODE_XXX` può essere:

- **FLASH_MODE_OFF**: in questo caso il flash è spento quando viene scattata la foto.
- **FLASH_MODE_ON**: al momento dello scatto il flash viene acceso per la durata dello scatto.
- **FLASH_MODE_AUTO**: `CameraX` si occupa automaticamente della scelta; in ambienti luminosi il flash resterà spento e in ambienti bui il flash verrà acceso.

4.3.10 Focus and Metering

Un'altra funzionalità fornita da `CameraX` è il controllo della messa a fuoco e della misurazione dell'esposizione. Questo viene attivato attraverso il metodo `startFocusAndMetering()` disponibile in `CameraControl`.

Utilizzando l'azione di messa a fuoco e misurazione (`FocusMeteringAction`) specificata, vengono impostate le regioni di messa a fuoco automatica (AF), misurazione dell'esposizione (AE) e bilanciamento del bianco (AWB) nel sensore della fotocamera. Questa funzione è spesso utilizzata per implementare la caratteristica "tap to focus" o "tocca per mettere a fuoco" in molte applicazioni fotografiche.

4.4 Pro e Contro

I pro e contro principali di `CameraX` sono:

4.4.1 Pro

- **Astrazione semplificata**: `CameraX` offre un'interfaccia API unificata che semplifica l'utilizzo della fotocamera nei dispositivi Android. Questo, come vedremo dopo, può essere considerato sia un vantaggio che un potenziale svantaggio.

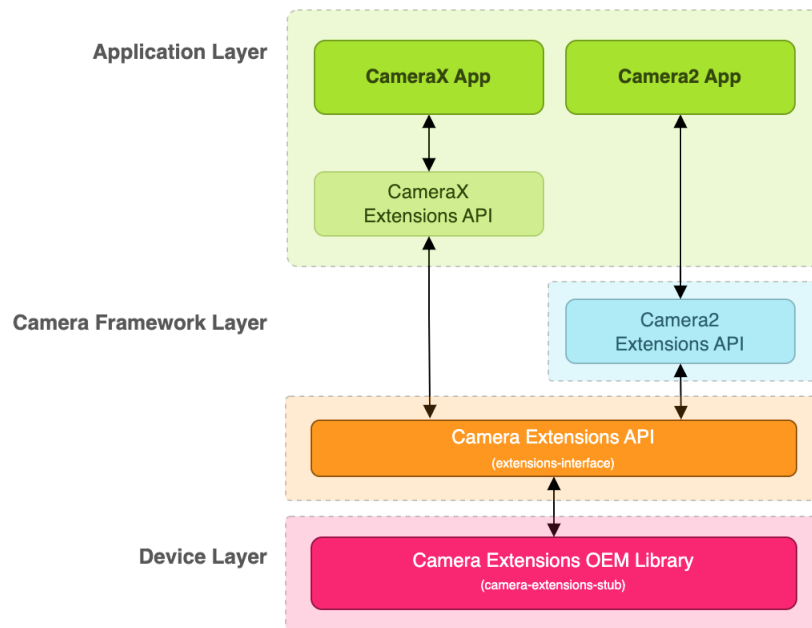


Figura 6: Struttura di cameraX

- **Supporto multi-dispositivo:** CameraX è progettata per funzionare su una vasta gamma di dispositivi Android, offrendo una grande compatibilità con la maggior parte degli smartphone e tablet. Questo semplifica lo sviluppo di applicazioni che utilizzano la fotocamera, eliminando le complicazioni legate alle differenze di utilizzo della fotocamera che variano a seconda del produttore del dispositivo.
- **Funzionalità avanzate:** la libreria fornisce un supporto completo per il controllo manuale dei parametri della fotocamera, come esposizione, ISO e messa a fuoco. Inoltre, offre la possibilità di catturare immagini nel formato RAW, ossia un formato di file per le immagini privo di compressione, che è in grado di conservare tutti i dati grezzi acquisiti dal sensore. Attraverso queste funzionalità CameraX permette di ottenere fotografie di alta qualità, rendendo le foto ottenute paragonabili a quelle dell'applicazione nativa del dispositivo.
- **Adattabilità alle modifiche:** l'astrazione di CameraX facilita l'aggiornamento delle funzionalità della fotocamera nel tempo. Se vi sono modifiche nell'accesso alla fotocamera o nuovi miglioramenti per il suo utilizzo, non è necessario che gli sviluppatori riscrivano il codice per usufruirne.

4.4.2 Contro

- **Astrazione limitante:** in alcuni casi il livello d'astrazione può portare, in caso di funzionalità specifiche, limitazioni o differenze di comportamento tra i diversi dispositivi. Alcuni dispositivi potrebbero non supportare tutte le funzionalità desiderate o potrebbero comportarsi in modo diverso rispetto ad altri.

- **Limitazioni della versione Android:** nonostante CameraX richieda una versione di Android uguale o superiore alla 5.0 (API 21), coprendo comunque circa il 98% dei dispositivi, gli eventuali dispositivi con una versione di Android inferiore non possono accedere a questa libreria e questo comporta che le applicazioni che vogliono supportare anche versioni precedenti devono utilizzare altre librerie specifiche per la fotocamera.
- **Overhead prestazionale:** l'utilizzo di un'astrazione come CameraX potrebbe introdurre un piccolo overhead sulle prestazioni, quindi una perdita di prestazioni, rispetto all'utilizzo di altre API per l'accesso alla camera. Questo dipende dal fatto che CameraX deve interpolarsi con il dispositivo per utilizzare la camera, aggiungendo un livello di complessità e richiedendo un maggiore utilizzo della CPU.

4.4.3 Conclusioni

Come già accennato in precedenza l'obiettivo di CameraX è quello di semplificare lo sviluppo delle funzionalità della fotocamera nelle applicazioni Android.

Essendo comunque una classe abbastanza recente, non ha ancora raggiunto il massimo delle potenzialità e presenta delle limitazioni nell'utilizzo di alcune implementazioni. Ha però delle ottime premesse e attualmente è la soluzione più semplice ed efficace, anche in termini di risultato, per utilizzare la fotocamera. Tuttavia, è importante considerare le attuali limitazioni di CameraX, specialmente per quanto riguarda l'implementazione di funzioni più complesse e il supporto a dispositivi più vecchi. Queste limitazioni potrebbero influire sulla scelta di utilizzare CameraX in determinati scenari più particolari.

Nonostante ciò, CameraX rappresenta comunque una soluzione conveniente e semplice per consentire l'utilizzo della fotocamera. È però importante considerare attentamente i trade-off tra facilità d'uso, funzionalità offerte e limitazioni presenti per determinare se CameraX sia la scelta appropriata per le proprie esigenze. In sintesi, nonostante CameraX abbia ancora spazio per migliorare, attualmente è una soluzione di facile utilizzo ed efficacia per l'accesso alla camera nelle applicazioni Android.

5 Conclusioni

In questo capitolo verrà fatto un confronto generale tra Camera, Camera2 e CameraX e verrà motivato il perché la scelta di sviluppo è ricaduta su CameraX.

5.1 Confronto generale

La classe Camera, ormai deprecata, fornisce un'interfaccia di basso livello, migliorata sia da Camera2 che da CameraX, e permette alle applicazioni che la usano di accedere alle funzioni base di una fotocamera (visualizzare un'anteprima, scattare foto, registrare video).

Come detto, l'API di Camera2 rende più flessibile l'utilizzo degli elementi della fotocamera. Essa si presenta come un'evoluzione di Camera, andando ad aggiungere altre funzionalità rispetto a quelle di quest'ultima; tra di esse si annoverano la burst capture o la cattura di video in HDR (al momento non ancora supportata da CameraX). Il numeroso insieme di properties accessibili di `CaptureRequest` amplia la lista dei `Camera.Parameters`, garantendo un supporto atto a personalizzare diversi parametri della fotocamera del dispositivo, come l'apertura del diaframma, il fattore di zoom o la compensazione dell'esposizione, e anche un controllo più approfondito sulla cattura delle immagini, attivando o disattivando opzioni come l'auto-framing o la modalità Zero Shutter Lag.

L'API di CameraX, a sua volta, offre un'interfaccia sensibilmente più semplice e ad alto livello rispetto a quella di Camera2. CameraX inoltre risolve problemi di compatibilità in modo trasparente, sollevando gli sviluppatori da oneri gravosi in cui potrebbero incorrere con Camera e Camera2. Un altro punto a favore di CameraX è certamente rappresentato dalla Extensions API, un modo più semplice ed accessibile per l'implementazione di modalità foto avanzate come le già citate Bokeh o Face Retouch. Le motivazioni che ci hanno convinto ad utilizzare CameraX per la realizzazione della nostra applicazione verranno trattate con più cura nella successiva sezione.

A discapito dei vantaggi in semplicità ed elasticità di CameraX, Camera2 può risultare la scelta giusta per chi intende sviluppare un'applicazione che permetta un controllo più avanzato e personalizzabile dei parametri della fotocamera o l'utilizzo della tecnologia HDR per la registrazione di video, fornendo quindi all'utente uno strumento più professionale e di uso meno saltuario.

5.2 Perché abbiamo scelto CameraX

Per la realizzazione della nostra applicazione abbiamo deciso di utilizzare la libreria di CameraX. Ci sono diversi motivi che ci hanno spinto a optare per questa libreria.

CameraX rende più semplice a noi sviluppatori l'utilizzo della camera all'interno delle applicazioni in quanto è la libreria a fare da tramite con l'hardware. Con CameraX non dobbiamo preoccuparci dei dettagli di implementazione specifici del dispositivo o delle complessità di Camera2. La libreria si occupa di gestire automaticamente la compatibilità con una vasta gamma di dispositivi Android, consentendoci di concentrarci sulle funzionalità desiderate senza dover affrontare complessità aggiuntive.

Il vantaggio che ci avrebbe portato Camera2 sarebbe stato un maggiore controllo sulla fotocamera e i suoi parametri, tuttavia, avrebbe richiesto una maggiore complessità di sviluppo. Quindi abbiamo scelto CameraX per semplificare l'utilizzo della fotocamera nella nostra app, pur offrendo un buon livello di controllo e funzionalità.

Inoltre è Google stessa a raccomandare agli sviluppatori di usare CameraX. Questo si può notare direttamente nelle pagine ufficiali di Android Developer, dove sia nella sezione relativa a Camera che a Camera2 è presente un banner che cita esplicitamente l'utilizzo di CameraX.

Camera:

```
1 "Note: This page refers to the Camera class, which is deprecated. We recommend using  
    CameraX or, for specific use cases, Camera2. Both CameraX and Camera2 support  
    Android 5.0 (API level 21) and higher".
```

Camera2:

```
1 "Note: This page refers to the Camera2 package. Unless your app requires specific, low-  
    level features from Camera2, we recommend using CameraX. Both CameraX and Camera2  
    support Android 5.0 (API level 21) and higher".
```

In sintesi, la scelta di CameraX per la nostra applicazione è stata motivata dalla raccomandazione di Google, dalla sua semplicità d'uso confrontato con Camera2 e dalle funzionalità che offre.

Riferimenti bibliografici

- [1] *Overview Camera*, <https://developer.android.com/training/camera-deprecated>
- [2] *Using an Intent to make a photo*, <https://www.vogella.com/tutorials/AndroidCamera/article.html>
- [3] *List of Camera.Parameters with description*, <http://www.dre.vanderbilt.edu/~schmidt/android/android-4.0/out/target/common/docs/doc-comment-check/reference/android/hardware/Camera.Parameters.html>
- [4] *Camera2 Version Support*, https://www.xda-developers.com/camera2-api/?newsletter_popup=1
- [5] *HDR videos*, <https://www.androidauthority.com/what-is-hdr-photography-1021421/>
- [6] *Multi-camera API*, https://www.youtube.com/watch?v=u38w0v2a_dA&ab_channel=AndroidDevelopers
- [7] *Esposizione e compensazione*, <https://fotocomefare.com/compensazione-dellesposizione/>
- [8] *Hot Pixel*, <https://www.lightpoint.info/it/foto/lezioni/12-hot-pixel-trucco-canon>
- [9] *CaptureRequest properties*, <https://developer.android.com/reference/kotlin/android/hardware/camera2/CaptureRequest>
- [10] *Apertura diaframma*, <https://fotocomefare.com/apertura-diaframmi-tutorial-fotografia/>
- [11] *Overview CameraX*, <https://developer.android.com/training/camerax>
- [12] *CameraX Versions*, <https://developer.android.com/jetpack/androidx/releases/camera>
- [13] *Camera Extensions*, <https://developer.android.com/training/camera/camera-extensions>
- [14] *An Introduction to CameraX*, <https://gabrieltanner.org/blog/android-camerax/>
- [15] *Android 13 migliorerà l'esperienza social e fotografica con questo strumento*, <https://www.tuttoandroid.net/news/2022/05/13/google-camerax-i-o-2022-a-cosa-serve-947906/>
- [16] *CameraX, Google lavora per migliorare l'esperienza fotografica di Android*, <https://www.hdblog.it/smartphone/articoli/n518850/google-fotocamera-frammentazione-android-camerax/>
- [17] *Android 13, passi avanti nell'uso della fotocamera in app terze*, <https://www.hdblog.it/google/articoli/n555779/android-fotocamera-terzi-video/>
- [18] *CameraX: HDR, Night and Portrait mode*, <https://himanshufi.medium.com/camerax-hdr-night-and-portrait-mode-655888aabbf28>

- [19] *Apply special effects to images with the CameraX Extensions API*, <https://medium.com/androiddevelopers/apply-special-effects-to-images-with-the-camerax-extensions-api-d1a169b803d3>
- [20] *What Is a Camera's Buffer?*, <https://photographylife.com/camera-buffer-explained#:~:text=What%20is%20the%20Camera%20Buffer%20Anyway%3F>
- [21] *CameraX: Make photography easier on Android!*, <https://www.codementor.io/blog/camerax-7fh6g0xyyg>
- [22] *What Is A RAW Photo?*, <https://blog.upskillist.com/what-is-a-raw-photo/>